Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №5
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»»**
Powershell terminal

Виконав:
Студент групи ІА-22
Парій І. Р.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Мета:
Ознайомитися з основними шаблонами проєктування, такими як «Adapter», «Builder», «Command», «Chain of Responsibility», «Prototype», дослідити їхні принципи роботи та навчитися використовувати для створення гнучкого та масштабованого програмного забезпечення.

Варіант
..8 Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server)
Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Хід роботи

1. Реалізувати не менше 3-х класів відповідно до обраної теми

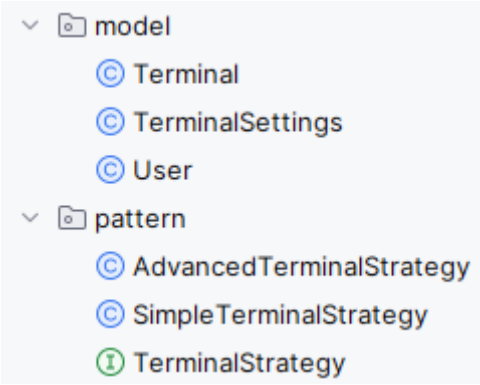

Рис. 1 — Структура проекту

У процесі виконання лабораторної роботи було створено 4 класи, які забезпечують реалізацію функціоналу PowerShell термінал (рис. 1). Наведемо детальний опис кожного з цих класів:

1. Інтерфейс TerminalCommand
2. Це інтерфейс, який визначає базовий контракт для всіх команд, що можуть виконуватися в терміналі. Він реалізує патерн Command, дозволяючи кожній команді мати метод execute(). Завдяки цьому інтерфейсу термінал може динамічно змінювати або виконувати команди без необхідності знати їх внутрішню реалізацію. Це забезпечує гнучкість та розширюваність для підтримки різноманітних функціональностей терміналу.
3. Клас ChangeColorCommand

   Реалізує зміну кольору інтерфейсу терміналу, наприклад, кольорів синтаксичних конструкцій або фону вікна. Ця команда дозволяє користувачам змінювати візуальні параметри терміналу, що є важливим для персоналізації роботи в PowerShell. Вона може бути використана як частина абстрактної фабрики для створення стилів або як окрема команда в системі.
4. Клас ResizeWindowCommand

   Відповідає за зміну розмірів вікна терміналу, задаючи його ширину та висоту. Ця команда забезпечує можливість адаптації терміналу до потреб користувача, дозволяючи працювати у вкладках, режимі розділення екрана або зручного масштабу вікна. Вона інтегрується в систему як частина налаштувань інтерфейсу, доповнюючи можливості зміни кольорів та інших параметрів.
5. Клас TerminalInvoker

   Це виконавець команд, що інкапсулює логіку вибору та запуску конкретних команд, реалізуючи патерн Command. TerminalInvoker забезпечує взаємодію між клієнтом (користувачем) і виконуваними командами. Завдяки цьому класу можна динамічно задавати різні команди, що дозволяє терміналу залишатися універсальним інструментом для виконання різноманітних завдань.

2. Реалізувати один з розглянутих шаблонів за обраною темою

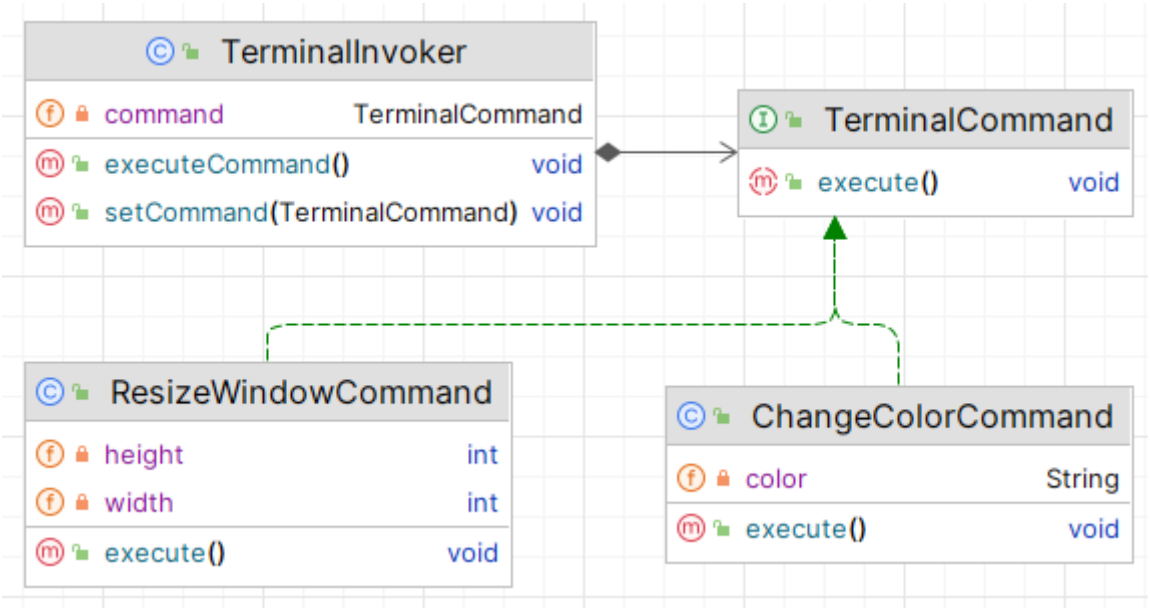

Рис. 2 — Діаграма класів

У контексті проєкту PowerShell терміналу, патерн Command реалізований для організації динамічного виконання різних операцій, пов’язаних із функціональністю терміналу, таких як налаштування інтерфейсу, зміна параметрів вікна та виконання команд PowerShell.

Як це реалізовано:

1. Інтерфейс TerminalCommand:
   - Він визначає єдиний метод execute(), який виконує команду. Це дозволяє кожній команді мати уніфікований спосіб виконання, незалежно від її специфіки.
2. Конкретні команди (ChangeColorCommand, ResizeWindowCommand):
   - Реалізують інтерфейс TerminalCommand і містять логіку для виконання специфічних операцій, таких як зміна кольору інтерфейсу або розмірів вікна.
3. Виконавець команд (TerminalInvoker):
   - Цей клас забезпечує збереження посилань на команди та їх виконання. Він діє як посередник між клієнтом і командами, дозволяючи змінювати команди динамічно в залежності від потреб.
4. Клієнт:
   - Користувач або інтерфейс терміналу створює конкретні команди та передає їх до TerminalInvoker, щоб виконати потрібну дію.

Проблеми, які вирішує:

1. Розділення відповідальності:
   - Логіка виконання команд інкапсульована в окремих класах, що спрощує підтримку та тестування коду.
2. Можливість динамічного виконання:
   - Користувач може легко змінювати або додавати команди без впливу на структуру виконавця або клієнта.
3. Розширюваність:
   - Додавання нових команд (наприклад, для зміни стилю вкладок або запуску PowerShell-скриптів) не потребує модифікації існуючих компонентів, що відповідає принципу відкритості/закритості (OCP).
4. Історія дій та скасування:
   - Патерн Command легко інтегрується з механізмами історії виконаних дій, що дозволяє реалізувати функціональність "скасування" (undo) для операцій, таких як зміна налаштувань.
5. Уніфікований підхід:
   - Використання єдиного інтерфейсу для всіх команд забезпечує послідовність і спрощує обробку різних типів операцій.

Переваги використання патерна Command:

1. Модульність:
   - Команди є незалежними від інших частин системи, що робить їх легко замінними або повторно використовуваними.
2. Інкапсуляція:
   - Логіка виконання ізольована в окремих класах, що зменшує складність основного коду терміналу.
3. Гнучкість:
   - Користувачі можуть налаштовувати та змінювати поведінку терміналу в реальному часі, додаючи нові команди або сценарії без перезапуску програми.
4. Легке тестування:
   - Кожна команда може бути протестована незалежно, що підвищує надійність системи.

Застосування

```
@PostMapping("/changeColor/{color}")
public String changeColor(@PathVariable String color) {
    TerminalCommand command = new ChangeColorCommand(color);
    invoker.setCommand(command);
    invoker.executeCommand();
    return "Color command executed";
}

@PostMapping("/resizeWindow/{width}/{height}")
public String resizeWindow(@PathVariable int width, @PathVariable int height) {
    TerminalCommand command = new ResizeWindowCommand(width, height);
    invoker.setCommand(command);
    invoker.executeCommand();
    return "Resize command executed";
}
```

Рис. 3 — Застосування патерну

У цьому коді використано патерн Command для обробки HTTP-запитів, що змінюють параметри терміналу. Метод changeColor створює команду ChangeColorCommand, яка відповідає за зміну кольору інтерфейсу, а метод resizeWindow створює команду ResizeWindowCommand для зміни розмірів вікна. Обидві команди передаються у виконавця invoker, який викликає їх метод execute(). Це дозволяє інкапсулювати логіку виконання команд і спрощує додавання нових дій, таких як інші налаштування чи функції терміналу, зберігаючи єдину структуру обробки.

Висновки:

Патерн Command у цьому проєкті дозволяє зберегти гнучкість, масштабованість і модульність терміналу PowerShell. Він полегшує обробку різних дій у терміналі, забезпечуючи зрозумілу структуру коду і розширюваність системи.